UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

MASTER COPY - FOR REPRODUCTION PURPOSES

(2)

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY AD-A209 133	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARO 25998.1-EL-SBI
6a. NAME OF PERFORMING ORGANIZATION Software Architecture & Eng. Inc.	6b. OFFICE SYMBOL (If applicable)
	7a. NAME OF MONITORING ORGANIZATION U. S. Army Research Office
6c. ADDRESS (City, State, and ZIP Code) 1600 Wilson Boulevard, Suite 500 Arlington, VA 22009-2403	7b. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U. S. Army Research Office	8b. OFFICE SYMBOL (If applicable)
	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAL03-88-C-0033
8c. ADDRESS (City, State, and ZIP Code) P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Software Technology to Support Real-Time Embedded Artificial Intelligence-Based Applications

12. PERSONAL AUTHOR(S)
Thomas E. Shields

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 10/1/88 TO 3/31/89	4 May 1989	22

16. SUPPLEMENTARY NOTATION
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Software Technology, Artificial Intelligence, Knowledge Based Systems, Real-Time System

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Current-generation KB systems run-time implementations are too abstract and non-deterministic for use in real-time systems. The real-time applications designer just cannot deal with systems that are non-deterministic and do not provide explicit control. The proposed research ideas were either abandoned or were not developed much beyond their initial conception. However, the fundamental conclusion of this research is that the proposed ideas were really attempting to tackle the wrong problems.

Keywords: artificial intelligence, knowledge based systems, operations research, computer architecture. (KR)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ARO 25998.1-EL-SB

SBIR PHASE I

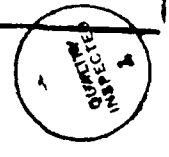
FINAL TECHNICAL REPORT

Software Technology to Support Real-Time Embedded Artificial Intelligence-Based Applications

Principle Investigator
Thomas E. Shields

Contract No. DAAL03-88-C-0033

CLIN No. 0002AB



4 May 1989

Prepared for:
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

Prepared by:
Software Architecture and Engineering, Inc.
1600 Wilson Blvd., Suite 500
Arlington, VA 22209

TABLE OF CONTENTS

1. INTRODUCTION

Artificial Intelligence (AI) technology promises to solve many knowledge-intensive problems that cannot be solved in conventional ways. Knowledge-based (KB) systems technology (also referred to, popularly, as "expert systems" technology) represents the first AI technology to be used with success in commercial and military applications, both in prototype and in some restricted operational environments. KB systems have been used extensively in the AI community for modeling "intelligent" behavior. Developers built early KB applications using specialized languages (e.g., Lisp) and special-purpose processors (e.g., Lisp machines). The first widely available KB applications were built using "expert system shell" tools, which allowed development of "rule bases" in a generalized, non-procedural English-like language format, and provided a generalized "inference engine" to interpret those rule bases. Large organizations, including various DoD agencies, are currently debating how to integrate successful KB application prototypes into operational and next-generation environments. Most KB systems are compute-intensive, and consequently operate relatively slowly. This has prevented the use of such applications in operational environments requiring high performance or real-time response.

One of the major influences on the requirements for next-generation real-time systems is the (perceived) need for KB capabilities. Next-generation real-time systems will continue to be developed for application domains similar to those of current-generation systems. However, these new systems are expected to be capable of exhibiting intelligent, adaptive, and highly dynamic behavior, and will probably have long system life-cycles. Failures in these new systems will have catastrophic consequences. This compounds the already difficult engineering problems faced in building current-generation real-time systems. As an example, advanced KB systems are characterized by their adaptability and complexity, making it effectively impossible to precalculate all possible combinations of tasks that might occur. This precludes use of static scheduling policies common in current-generation real-time systems [37].

One of the primary goals of introducing KB subsystems into embedded real-time (weapons) systems applications is to maintain, and hopefully improve, the "situation awareness" of the human component of the system by reducing the human's detail-level workload, enabling him to concentrate on more global tasks. Many of the assorted independent details of operating a weapon system, currently dealt with by the human, would be integrated by an KB subsystem, and perhaps

summarized for the human on demand or continuously. Examples of the types of tasks which have been suggested include:

- intelligent navigation;
- mission planning and dynamic re-planning;
- intelligent internal system monitoring;
- emergency procedure aids;
- standard procedure monitoring;
- threat assessment; and
- tactical consultation.

The thesis of this SBIR project was that the constraints of real-time applications cannot be satisfied simply by choosing a faster execution environment or a more efficient implementation of "classical" KB techniques, as have been the primary approaches to date. The use of generalized KB structures and reasoning paradigms cannot meet real-time performance requirements, especially on conventional, standard military hardware. Real-time KB applications must be explicitly analyzed, designed and built for real-time performance, as opposed to attempting to migrate existing implementation strategies into the real-time environment.

This research project produced no results warranting the preparation of a SBIR Phase II proposal. The ideas contained in the proposal were not proven to be useful for various reasons; the fundamental drawback was ultimately recognized to be a lack of proper understanding by the author of the research area at the time the proposal was prepared. Consequently, this project ended up as "pure research" (emphasis on the "re" prefix), with no significant original idea development. However, with the better understanding and appreciation of the research area of real-time AI that this project enabled me to develop, directions for future investigation have been identified that should prove more productive.

Section 2 contains a general review of the "state-of-the-industrial-art" of real-time, AI, and real-time AI systems; Section 3 contains a synopsis of the proposed research program; Section 4 discusses the results, such as they are, of the research project; and Section 5 presents a synopsis of directions for future investigation in the area of real-time AI systems.

2. ANALYSIS

Over the past few years, especially since the initiation of DARPA's Strategic Computing Initiative, there has been a growing assumption that "real-time AI systems" will be able to do all sorts of wonderful things for military weapons systems. To date, the development of these systems seems to be taking the "traditional" AI ad-hoc approach, wherein a system is built, tried out, discovered to not work in real time except for toy problems, and then another system is built. The "lessons learned" reported from these prototype projects seem to be, in general, that either faster computers, or parallel processing, or some combination of both, will solve the real-time performance problems. There does not seem to be much work being done in examining the fundamental algorithms being used to build these prototypes [23].

2.1. Real-Time Application Technology

A real-time application must not only perform the right action, but must do it on time and often within severe time constraints. In the real-time environment, in general, a correct answer is wrong if it is given too late. There is often a distinction made between "soft" and "hard" real-time systems. The main distinction seems to be that "soft real-time" applications accept that some tasks will not always meet the timing constraints, and attempt to ensure that those tasks that do miss are the lowest priority tasks. "Hard real-time" applications admit no such leeway. This means that the real-time application implementor needs to know exactly how the software system works, and how long it takes to perform a given task. Deterministic behavior is an essential requirement of an embedded real-time system. Random timing failures may lead to catastrophic situations. To meet tight timing requirements, current practice in real-time programming relies heavily on manual machine-level optimization techniques. These techniques are labor intensive and tend to introduce internal instruction sequence timing assumptions on which the correctness of an implementation depends. Reliance on clever hand-coding and implicit timing assumptions is a major source of bugs in real-time programming. A primary objective in real-time-systems research is to automate the synthesis of highly efficient code and customized resource schedulers from real-time performance specifications by exploiting sophisticated optimization techniques and scheduling theory [37].

2.2. KB Application Technology

Most work in the development of KB inferencing technology has been devoted to knowledge representation schemes and inference algorithms that provide simpler, easier to use, or more accurate techniques for modeling a problem domain. The issue of efficiency has been primarily related to computational feasibility. These algorithms have been designed, for the most part, to deliver a "near optimal" response, given the extent of knowledge provided, with performance a secondary consideration. The performance bottleneck has traditionally been addressed by manual local optimization and by resorting to lower level implementation languages that more closely map to the underlying hardware. These approaches are not desirable because of high development costs and portability considerations. Another approach has been to develop specialized hardware that more closely maps to the application functionality. This approach is not desirable, at least in the short run, due to the military's requirement for standard hardware. Furthermore, this approach will not solve the fundamental problem in the long run. The short history of computing demonstrates, as has been noted by many, including [37], that the availability of increased computing functionality results mostly in demand for applications requiring even greater functionality.

KB technology offers a body of techniques for the manipulation and use of symbols representing "knowledge". Many of these techniques have been encapsulated into software tool environments, referred to as "expert system shells". While these tools are extremely useful for rapid prototyping of KB applications, they have flaws rendering them unsuitable for real-time processing.

Existing expert system tools contain a generic facility that performs symbol manipulations, referred to as the "inference engine". These facilities operate on the domain knowledge, which is represented by instantiations of generic data structures, in an interpretive fashion at run-time. The consequences are that:

(1) the domain knowledge consumes more memory than absolutely necessary due to use of generic data structures,

(2) computation takes longer than necessary due to the interpretive implementation, and

(3) the symbol manipulation facilities are not optimal, in general, for a particular knowledge base due to the generic nature of the implementation.

A second problem with existing expert system tools is that their reasoning is inflexible and unresponsive in a dynamic environment. Many tools rely exclusively on production-rule-based reasoning, an approach which has many limitations both in the maintenance phase and at run-time, in addition to its advantages during initial development. Existing expert system tools also have no knowledge of history, and so are doomed to recommend the same response over and over again. Even if a response is "near optimal", repetitious behavior may fail in a tactical environment against a competing opponent. Intelligent tactical systems must retain knowledge of previous actions and weigh current decisions in light of this memory. This lesson has already been learned in attempts to build intelligent chess programs. In a series of matches against a single opponent, programs with superior ratings will tend to lose consistently to humans who detect a predictability to the program's behavior.

The "classical" KB inferencing techniques which can be used to automate knowledge-intensive tasks are in the class of "hard" problems referred to by mathematicians as "NP complete." Problems in this class are all characterized by having a time-order complexity that is exponential in the number of components of the problem that must be considered. Solutions to these problems, in the general case, involve exhaustive enumeration of all potential solutions. In general, these computations have potentially unbounded requirements for processor cycles and memory.

The classical formulation of these algorithms is as a search of a decision tree. Unfortunately, it is easy to specify knowledge in ways that lead to infinite search sequences. The classical solutions to this problem have involved techniques of using knowledge to prune the search tree dynamically at run-time (selective search), manual reprogramming by rewriting the knowledge specification, or alternative search strategies, such as forward (data driven) versus backward (goal driven) and breadth-first (parallel evaluation) versus depth-first (sequential evaluation) strategies. While these techniques may work for a particular subset of problems, none provide general solutions: selective search may miss the solution; forward and breadth-first search strategies may result in computation and storage of irrelevant information; and reprogramming may be arbitrarily difficult for certain knowledge and may result in the implicit embedding of control information to the detriment of future knowledge specification maintenance.

Many workers in the field have begun to explore the possibility of using parallel processing techniques to minimize the elapsed time computational costs. However, it

has been shown [23] that the maximum speedup due to parallel processing, in the ideal case, is absolutely bounded by the number of processors which a problem can effectively support, so that distributing an intractable (exponential time-order complexity) problem over many processors is unlikely, in general, to result in a tractable (polynomial time-order complexity) problem.

2.3. Real-Time KB Application Technology

Real-time AI research currently emphasizes reasoning about time-constrained processes and using heuristic knowledge to control or schedule these processes [37]. The fundamental requirement to be addressed is the development of application algorithms that can guarantee satisfaction of resource constraints, while also delivering acceptable answers. The obvious constraint is, of course, that a response must be produced within some time interval that may not be known in advance, although a worst-case value will either be known or assumed. Time, however, is not the only constrained resource. Such resources as memory and external devices may be scarce in general or may become scarce. An essential aspect of the problem is that, in typical real-time applications, there are many *independent* processes which interact or compete with each other for the same pool of resources. These issues are generally applicable to any real-time application. However, the non-deterministic nature of "classical" AI algorithms makes these requirements particularly difficult. For example, the dynamic nature of symbolic systems has lead to development of implementation approaches that require automated memory management. Current garbage collection techniques make it difficult, to guarantee a maximum system latency [37], although there has been some work done in an attempt to alleviate this difficulty [12]. Furthermore, symbolic systems are typified by opportunistic control strategies that provide the ability to dynamically direct program execution in response to (real-time) events. This contrasts sharply with the current real-time systems assumption of a statically determined "decision tree" control sequence. The implication here is that the sequences of processes can not be predetermined [37].

3. PROPOSED APPROACH

In the proposal, I state that real-time KB system problems must be approached from a software engineering perspective, which is slightly different from that found in the literature or as presented by many current government-sponsored projects. The use of generalized KB structures and reasoning paradigms cannot, in general, meet real-time performance requirements, especially on conventional, standard military computer systems. A change in emphasis for developing real-time KB systems is needed. The generalized symbol manipulation facilities must be abandoned in favor of operations which are specialized to the particular domain knowledge itself, avoiding costly run-time generality.

3.1. Phase I Technical Objectives

The scope of the proposed Phase I investigation was to explore the feasibility of possible software architectures and strategies for implementation of real-time symbolic computations. The class of computations to be addressed were those which are implemented as search-intensive, NP-complete algorithms. The necessity to use algorithms of this type raises the question of whether real-time performance can ever be reliably realized by KB applications.

The objectives of the Phase I work were to:

(1) Investigate the potential for use of constrained versions of the "classical" KB paradigms (in tactical weapons systems).

(2) Investigate the potential for speedup of KB paradigms on conventional hardware architectures through the application of the "compilation paradigm". The essence of this approach is to generate specialized instantiations of generic data structures and interpretive algorithms that are tuned to a particular (KB) problem.

3.2. Constrained Inference

I suggested two possible approaches to the development of "constrained" versions of classical KB paradigms. The

essential question that I hoped this research might shed some light on was whether, assuming that these ideas had any merit, such constrained paradigms could be scaled to perform realistic and useful problem solving. The first suggestion drew on an analogy with results in Operations Research (OR) research. Many of the underlying algorithms that are used in the solution of KB problems appear to be very similar to algorithms that have been developed in the field of OR. The idea was that examination of approaches to efficient problem solving through imposition of constraints on the problem space, as has been done in the field of OR, might lead to similar results with KB problems. Another approach suggested for some problems was the utilization of "feedback" techniques, such as has been done in the field of robotics. Assuming that a simplified model of the problem can be developed, it is possible to arrive at an approximate solution quickly, take initial action based this approximate solution, and then monitor the success of that solution while possibly incrementally adjusting it.

3.3. The Compilation Paradigm

I suggested application of what I term the "compilation paradigm" as a means to achieve more efficient implementation of classical KB technology. This suggestion was based on the following analogy. In time critical emergency situations, humans tend to rely on "standard" procedures, rather than on inferring an appropriate course of action from an examination of some "deep knowledge" of the world. Such procedures can be invoked quickly, with little or no thinking, in a properly trained human. In essence, the "thinking" was done in advance, or by someone else and learned by rote. This is analogous to the difference between an interpreted and a compiled implementation of a computer program. An interpreter chooses the particular sequence of hardware instructions to be executed for each program dynamically at program run-time, while a compiler allocates the necessary hardware instructions once in advance of program run-time.

An interpreter carries around a complete model of the semantics of the programming language it implements. A compiler also contains a complete model of the semantics of the programming language. However, in this case, the execution of a particular program is planned in advance of the actual execution. The program is analyzed and a specialized implementation containing only those hardware instructions necessary is generated. The result is faster execution of the program because of the elimination if the dynamic instruction selection. Furthermore, there is plenty

of time to perform extensive static analyses in order to infer special case constraints met by the program at execution time that permit selection of more efficient execution plans.

A generalization of the idea of "compiling a program" leads to the following notion of a "compilation paradigm":

(1) Given a particular instance of some high level abstract or non-procedural computational specification, perform an analysis of that specification to determine its particular execution requirements prior to its actual execution.

(2) From a sufficiently detailed analysis, a specialized execution plan can be generated in a lower level, less abstract, more procedural, more detailed specification language.

(3) If the output of this "compilation" is not yet at the level of the hardware on which the computation is to be executed, repeat at step (1) for this new computational specification.

(4) (Optimization) A more extensive and detailed analysis will permit a more specialized and efficient execution plan; for example, *exploiting* parallelism.

These ideas are loosely based on work done during the 1970s on compilation of very high level languages (e.g., see [34]).

An expert system consists of a specification of a knowledge base, together with an inference engine that interprets that knowledge base in the context of external input data, to arrive at some external output data. In essence, then, the knowledge specification is an abstract computational specification, or "program," that "executes" on the virtual machine implemented by the inference engine. Application of the compilation paradigm would transform a particular knowledge specification into an execution plan for the inference engine's virtual machine. Since this virtual machine is not directly implemented by current generation computer hardware, at least another application of the compilation paradigm is needed to reach an execution plan for a conventional machine architecture.

4. RESEARCH RESULTS

This section attempts to review the results of the research project. In general, the proposed research directions did not yield any useful new ideas. As I discovered during the project, my original understanding of the area of real-time and KB systems proved to be naive. I discovered that I was trying to address what amounted to the wrong set of fundamental problems. I have since come to the realization that the real solutions to development of successful real-time KB applications are not going to be found by pursuing versions of existing KB algorithms that run faster. Instead, new algorithms are needed that are based from their initial conception on principles of conservative use of resources rather than on extravagance, and new approaches to knowledge engineering are needed that are oriented toward providing quick approximate solutions which can be incrementally improved.

Current generation "state-of-the-industrial-art" KB technology is characterized by:

(1) standalone processing, even in those cases where a KB subsystem has been embedded within another application,

(2) human interaction orientation ("man-in-the-loop"),

(3) single-thread, synchronous processing,

(4) a "timesharing" mentality (resources can be waited for), and

(5) exhaustive analysis, with heuristic "tricks" to avoid complete enumeration where possible.

Any one of these characteristics causes a problem for incorporation into real-time systems. The totality makes it extremely difficult, if not impossible, to utilize KB technology as it stands today int hard real-time systems.

4.1. Operations Research

I proposed to perform a survey of the OR literature in an attempt to identify constrained problem solving paradigms that are analogous to classical KB problem solving paradigms, and then to explore the applicability of those approaches to the reformulation of typical KB problems to yield more efficient algorithms. This proposal was made without having first done more than a cursory review of

survey-level OR literature. With a somewhat more in-depth survey, we came to the realization that the type of problems addressed with KB technology are fundamentally different from the problems usually considered in areas of OR. In most OR problems, there is a fixed system having completely specified and static characteristics. The goal is to find optimal static solutions that minimize or maximize some attributes of the solution. Most real KB applications are highly dynamic, requiring on-line, adaptive algorithms. As has been mentioned earlier, current algorithms are based on heuristics, since complete, deterministic algorithms, where known, are NP-complete. This line of research was dropped, once the incommensurable problem characteristics were fully appreciated.

4.2. Incremental Approximations

One of the solutions often proposed to real-time performance problems is to use more powerful processors, or massively parallel processor combinations, which are becoming increasingly available. Unfortunately, a review of the history of computing demonstrates that the demand for more computing power has always outstripped the supply. If the past is any indications of what will occur in the future, the availability of more computing power will only result in new real-time applications, requiring greater functionality, possibly making the timing problems even worse. There is no substitute for intelligent allocation of finite resources [37].

In [20], an approach to real-time KB processing is described that requires that the system's control component reason about its overall objective relative to its current state, where the objective defines the criteria for acceptable solutions. This research has concentrated on incorporation of approximate reasoning into the planning facilities of a problem solver's control component, with the goal of efficiently constructing acceptable solutions within time constraints. This is the only published work I was able to find that even dealt with the ideas of incremental approximations in any depth. However, the approach of making a rough pass at solving the problem and then using the remaining time to incrementally refine the solution was abandoned in this published work due to the claim that refining parts of a solution and then propagating the changes throughout the solution state to arrive at the new solution was too costly.

This paper ([20]) makes a good case for the incorporation of domain or problem specific alternative

solution strategies which approximate the "optimal" solution stragegy, and point out that the approach does not obviate the incremental refinement approach as an additional aspect of problem solving. However, this approach requires that the problem solver be able to perform temporal reasoning in the problem domain to decide when to switch to approximate processing and which approximation to pursue, which requires a sophisticated control component that also needs to manage its own "solution" to the control problem (introspection). This appears to be a viable approach, but it is clearly a long way from being useable in hard real-time environments.

4.3. Exploitation of Parallelism

There are a number of aspects of typical KB systems that limit the amount of potential parallelism. One of the reasons for using KB technology in the first place is to make use of the KB application paradigm, wherein large amounts of "knowledge" are used to determine each action of the application. An individual "knowledge source" (e.g., a rule) typically does very little "processing" by itself when activated. In typical existing KB applications, only a very small percentage of the "working memory" will change during the knowledge activation step of each cycle of the inference algorithm. Potential parallelism could be increased by performing more "processing" within each knowledge source's activation, but this would also undermine the whole point (and power) of the paradigm. Another aspect of the problem is that knowledge bases typically must contain a wide range of "knowledge" in order to solve "interesting" tasks. The various knowledge sources in the system relate to different situations that may arise. The small number of changes to "working memory" at any cycle of the inference algorithm, as mentioned above, will not relate to more than a few of the knowledge sources. Hence, there is very little potential for increasing the exploitable parallelism with current KB technology.

4.4. KB Application Architecture

Thanks to the now classical architecture of KB systems -- separate working memory (data, or facts), knowledge base, and inference engine -- the KB application designer is able to concentrate on developing, in the knowledge sources within the knowledge base, the different relations and constraints of the problem, while leaving to the inference engine the task of linking these relations in order to

produce the results from data stored at a given moment in the working memory. This is an early example of a software development principle now referred to as "information hiding". A major advantage, from the point of view of KB application development, of this structure is that a KB system transfers responsibility for execution sequence control from the programmer to the KB system. Each knowledge source is regarded, in effect, as an individual "program" to be called up when needed, based on changes to data in the working memory. In conventional programs, subroutine structure and control flow must be designed explicitly to accommodate all potential operational situations. In contrast, control flow in KB systems is fluid; execution sequences may not be precisely known in advance. Many of the concepts behind this system architecture come from the time-sharing environment that was common during the early AI research projects, where one can wait for resources, where requests for resources can be resubmitted and where collection of "garbage" is desirable. This environment is diametrically opposed to the perspective of the real-time designer, where the software and system properties must be fixed, reliable and known ahead of time [24].

The strict separation (i.e., information hiding) of control, knowledge, and data that is the basis of the classical KB systems architecture represents a set of software design decisions that are optimized in ways that do not readily fit the needs of real-time application developers. The notable problem areas include the following:

(1) KB systems architecture has evolved, especially in the expert system shell tools, to the point where the control component is "standard", and consequently non-configurable by the application developer. This often results, for real applications, in the developer having to resort to "tricks" in the implementation of the knowledge sources and data in order to surreptitiously influence control decisions made by the inference engine. In effect, the technology as evolved in such a specialized way that it has reached the point where it is more often than not too restrictive. If a particular problem doesn't exactly fit the inference engine's control regime, the tool ends up getting in the way more than it helps in implementation of the application.

(2) KB systems technology has evolved through applications designed to respond with the best possible solution derivable from the knowledge base. This has resulted in development of what are basically exhaustive search solution strategies, with subsequent development of

heuristic search strategies which attempt to limit the search to profitable paths. The whole strategy of discovering the "best" solution through a search algorithm is a monolithic, "all or nothing" approach to the problem. What is really needed in the real-time environment is approximate and incremental multi-agent approaches.

(3) Typical KB applications are "man-in-the-loop" applications that respond to an initial stimulus from the human, and then run until the solution has been reached, requesting additional input (e.g., external data) when deemed necessary by the inference engine. This synchronous nature of the inference algorithm is perhaps the major stumbling block to utilization of KB technology in real-time systems. Real-time systems need to be able to deal with asynchronous events, and, more importantly, asynchronous events need to be able to immediately affect the control flow of the application. Some current expert system shell tools have attempted to allow the application developer to model asynchronous events, through such techniques as "daemons", but the underlying implementation is still synchronous, only dealing with the "asynchronous events" at well-defined points in the inference cycle. The major issue that needs to be addressed here is that an asynchronous event may need to redirect the current search path of the inference process before it has completed. This would require fairly major changes to current inference algorithm designs. In general, current KB technology is based upon centralized, synchronous control, whereas real-time KB technology needs to be able to utilize decentralized asynchronous control.

(4) Current KB technology is heavily oriented to interactive development and use. Major features of importance in current-generation expert system shell tools are those oriented toward explanation and justification of questions and answers. All of this is just excess baggage in most real-time systems, where there will either be no human interaction, or else no time to waste on such frivolous tasks. Redesign of run-time knowledge representation data structures and inference algorithms omitting these requirements may help solve some of the run-time memory requirements of KB applications.

4.5. Classification vs. Constructive Reasoning

Two general classes of problems have been addressed by KB applications. The first class of problems are those that utilize what has been termed "classification reasoning" [8]. Problems successfully approached with classification reasoning are those in which all solutions can be feasibly exhaustively enumerated in advance. These are typically well-defined problems from which a single answer must be selected. These problems involve minimal search, and often rely on an exhaustive search through a well-defined space. Most of the more well known KB systems (e.g., Mycin, Sacon, Sophie) fall into this category. This class of problems is generally what is solvable using the expert system shell tools currently found on the commercial market.

The second class of problems are those in which the solution space is much too large to exhaustively enumerate. Therefore, candidate solutions must be generated and evaluated at run-time. These types of problems are often game playing type scenarios (e.g, chess or command and control). Here, efficient generation and evaluation of possible solutions is of paramount importance. This is what is referred to as heuristic search in the literature. These types of systems perform what has been referred to as "constructive reasoning" [8]. Constructive *reasoning* requires larger knowledge bases and more difficult software development efforts. It is likely that any truly useful tactical real-time problems requiring KB solution techniques will fall into this class, which does not bode well for any short term successes, as there have not been many successful industrial or DoD applications using this paradigm; Xcon being the prominent exception. There have been some attempts to develop efficient "real-time" versions of classical heuristic search algorithms, (e.g., [15]), but these have not as yet been tried in real applications.

4.6. Compilation Paradigm

My ideas on what I term the "compilation paradigm" still seem viable, although I was not able to make any progress in developing them. Several references, notably [9] and [26,27,29], have had some successes in applying related ideas to the construction of more efficient implementations of particular AI algorithms. The main difference between what I was attempting to do, and what these other efforts were doing is that they *developed* a specific approach to a specific AI algorithm, while I tried to get my arms around a more general approach to the class

of KB algorithms. I eventually recognized that there is no straightforward approach here, although I suspect that I can eventually prevail by changing my tactics. My naive assumption was that there was an underlying, well-defined common representation in which any inference technique could be described. This may be true, but I was not able to identify it during the performance period of this contract.

The tactic that I should have used from the start on this project was to pick a particular instance of some inference technique, together with several realistic non-trivial KB applications built using that instance, and investigate the "compilation paradigm" approach within that more restricted context. I did, in fact, decide to switch to this approach near the middle of the performance period, but not long after I also had begun to realize that I was not addressing the real problem. At that time, I chose to try to define what the research direction really should be rather that pursue my original ideas. My thoughts on this are presented in Section 5 below.

I think that, given inference algorithms better suited to real-time problem solving, this approach has merit in realizing an efficient implementation. What will be required in order to make any progress with this idea is a way to define the behavior of an inference engine in terms of a set of well-defined primitive operations, rather than in terms of a high-level patterm match operation. This mechanism would allow the behavior of the inference engine to be specified in relation to a collection of particular knowledge sources (e.g., rules) on which it is supposed to operate. In other words, this would allow the semantics of the knowledge source "language" to be defined with respect to a particular inference engine. The main difference between this situation and traditional programming language compilation is that, in general, knowledge sources define non-deterministic behavior. One of the goals of this approach would then be to identify the deterministic sub-behavior, for which straight-line code could be generated, and encapsulate those sub-execution sequences within a control mechanism to provide the non-deterministic simulation, such as a multi-tasking real-time executive.

5. CONCLUSION

Current-generation KB systems run-time implementations are too abstract and non-deterministic for use in real-time systems. The real-time applications designer just cannot deal with systems that are non-deterministic and do not provide explicit control. As discussed in the preceding section, the proposed research ideas were either abandoned or were not developed much beyond their initial conception. However, the fundamental conclusion of this research is that the proposed ideas were really attempting to tackle the wrong problems.

The very essence of a real-time system is the requirement for a deterministic response to an external event. However, this does not explicitly include a requirement for a fast response. In fact, many process control applications only require response times on the order of a second, and many electromechanical systems only require response times on the order of a millisecond, neither of which comes close to taxing the response time of current generation microprocessors. The objective of fast computing is to minimize the average response time of a given set of tasks. However, the objective of real-time computing is to meet the individual timing requirement of each task. Rather than being fast (which is a relative term anyway), the most important property of a real-time system should be predictability; that is, its functional and timing behavior should be as deterministic as necessary to satisfy system specifications. Fast computing is helpful in meeting stringent timing specifications, but fast computing alone does not guarantee predictability [37].

Typical real-time application development attempts to substitute speed for predictability (and comprehensibility), on the theory that if everything can be done quickly enough, it will get done in time. This strategy has several shortcomings. First of all, it fails to explicitly define the behavioral specifications of the system. Without an understanding of these specifications, maintenance and evolution become increasingly difficult. Furthermore, it's virtually impossible to predict how a change in one aspect of the system's behavior will affect other aspects.

Another shortcoming is that this strategy provides only one method for satisfying behavioral specifications: speed up the components of the program. As a result, satisfying specific behavioral specifications becomes a matter of "ad hoc" experimentation rather than systematic engineering. Because of the complex interactions between components in a real-time system, this method provides no guarantee that the behavioral requirements will continue to be satisfied over

the life-cycle of an application.

This suggests that research directions in the area of real-time AI should not stress speedups of current-generation inference algorithms, contrary to what is suggested in [17]. While speed is important, we will never achieve any real results if this is the main thrust of the effort.

Instead, the following issues seem to be more fundamentally important, and suggest that alternate approaches to KB architectures and knowledge engineering need to be developed:

(1) multi-agent, asynchronous inference algorithms and architectures: The "black-board" architecture is a relatively recent development that holds promise for better integration into the real-time environment. This approach is characterized by multiple cooperating agents, rather than a single agent as is present for most inference engines. This architecture was used in the work reported in [20], and the RTEX system reported in [10] utilizes a similar architectural style.

(2) incremental, approximate reasoning algorithms: This must be accompanied by knowledge engineering methodologies that provide the necessary knowledge structures to support use of such algorithms. Current methodologies are oriented to the monolithic approaches used by expert system shell tools.

(3) inference algorithms exhibiting static or bounded memory requirements: For example, current backward chaining inference algorithms may follow unbounded paths through the knowledge base, and realizations of these algorithms tend to use recursion as the implementation strategy, both of which consume memory resources.

These issues need to be addressed first within specific problem contexts, as suggested in [20], prior to attempting to generalize the approaches. It may well be that there are not any domain- or problem- independent approaches to the development of real-time KB systems, but it seems clear that the issues are too complex to solve in the abstract before they have been solved for specific cases.

Appendix 1 Bibliography

(1) Bahnij, R., "A Fighter Pilot's Intelligent Aide for Tactical Mission Planning," DTIC/NTIS AD-A163 944, AFIT/GCS/ENG/85D-1, December, 1985.

(2) Barr, A., and Feigenbaum, E., The Handbook of Artificial Intelligence, Volumes 1, 2, William Kaufmann, 1982.

(3) Bonasso, R., "What AI Can Do for Battle Management: A Report on the first AAAI Workshop on AI Applications to Battle Management," AI Magazine, Vol. 9, No. 3, Fall 1988, pp. 77-83.

(4) Brintzenhoff, A., Christensen, S., Mangan, J., and Greco, J., "The Use of Ada Concurrent Processing Features in an Implementation of Parallel Tree Searching Algorithms," Proceedings of AIDA-87, Third Annual Conference on Artificial Intelligence and Ada, 13-15 October 1987, pp. 138-154.

(5) Burns, A., Lister, A., and Wellings, A., A Review of Ada Tasking, Lecture Notes in Computer Science, Vol. 262, Springer-Verlag, 1987.

(6) Butler, C., Hodil, E., and Richardson, G., "Building Knowledge-Based Systems with Procedural Languages," IEEE Expert, Vol. 3, No. 2, Summer 1988, pp. 47-59.

(7) Chien, Y., and Liebowitz, J., "Expert Systems in the SDI Environment," IEEE Computer, Vol. 19, No. 7, July 1986, pp. 115-122.

(8) Clancey, W., "Heuristic Classification," Artificial Intelligence, Vol. 27, No. 3, December, 1985, pp. 289-350.

(9) Collins, W., and Slothouber, L., "Expert System Control in Ada," Proceedings of AIDA-88, Fourth Annual Conference on Artificial Intelligence and Ada, 15-16 November 1988, pp. 17.1-17.11.

(10) De Feyter, A., "RTEX: An Industrial Real-Time Expert System Shell," Proceedings of AIDA-88, Fourth Annual Conference on Artificial Intelligence and Ada, 15-16 November 1988, pp. 6.1-6.22.

(11) Faulk, S., and Parnas, D., "On Synchronization in Hard-Real-Time Systems," Communications of the ACM, Vol. 31, No. 3, March 1988, pp. 274-287.

(12) Ford, R., "Concurrent Algorithms for Real-Time Memory Management," IEEE Software, Vol. 5, No. 5, September 1988, pp. 10-23.

(13) Hall, P., "Equivalence Between AND/OR Graphs and Context-Free Grammars," Communications of the ACM, Vol. 16, No. 7, July 1973, pp. 444-445.

(14) Hardin, D., and Albin, K., "Taking Inference to Task," Proceedings of AIDA-88, Fourth Annual Conference on Artificial Intelligence and Ada, 15-16 November 1988, pp. 8.1-8.9.

(15) Korf, R., "Real-Time Heuristic Search: First Results," in Proceedings of AAAI '87, Sixth National Conference on Artificial Intelligence, 13-17 July, 1987, pp. 133-138.

(16) Kramer, J., "AI Applications, Real-time Systems, and Ada," keynote address, Proceedings of AIDA-88, Fourth Annual Conference on Artificial Intelligence and Ada, 15-16 November 1988, pp. 2.1-2.12.

(17) Laffey, T., Cox, P., Schmidt, J., Kao, S., and Read, J., "Real-Time Knowledge-Based Systems," AI Magazine, Vol. 9, No. 1, Spring 1988, pp. 27-45.

(18) Leinweber, D., and Gidwani, K., "Real-Time Expert System Development Techniques and Applications," in Proceedings of Westex-86, IEEE Western Conference on Knowledge-Based Engineering and Expert Systems, 24-26 June, 1986, pp. 69-77.

(19) Leinweber, D., "Expert Systems in Space," IEEE Expert, Vol. 2, No. 1, Spring 1987, pp. 26-36.

(20) Lesser, V., Pavlin, J., and Durfee, E., "Approximate Processing in Real-Time Problem Solving," AI Magazine, Vol. 9, No. 1, Spring 1988, pp. 49-61.

(21) Liu, J., and Lin, K., "On Means to Provide Flexibility in Scheduling," Proceedings of the Second International Workshop on Real-Time Ada Issues, Ada Letters, Vol. VIII, No. 7, Fall 1988, pp. 32-34.

(22) Martin, J., "A Development Tool for Real-Time Expert Systems: CHRONOS," AlsyNews, March 1989, pp. 13-16.

(23) Norman, D., "Reasoning in Real-Time for the Pilot Associate: An Examination of a Model Based Approach to Reasoning in Real-Time for Artificial Intelligence Systems Using a Distributed Architecture," DTIC/NTIS AD-A163 947, AFIT/GCS/ENBG/85D-12, December, 1985.

(24) Parrish, L., "Running in Real-Time: A Problem for Ada," Defense Computing, September-October 1988, pp. 38-40.

(25) Pearl, J., Heuristics - Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, 1984.

(26) Reeker, L., "A Graphic Approach to Parallel Tasking and Data Abstraction for Symbolic Processing Applications in Ada," Proceedings of AIDA-86, Second Annual Conference on Artificial Intelligence and Ada, 12 November 1986, pp. 2.1-2.10.

(27) Reeker, L., and Wauchope, K., "Pattern-Directed Processing in Ada," Proceedings of the Second IEEE International Conference on Ada Applications and Environments, 8-10 April, 1986, pp. 49-56.

(28) Reeker, L., Kreuter, J., and Wauchope, K., "Artificial Intelligence in Ada: Pattern Directed Processing," technical report AFHRL-TR-85-12, May 1985.

(29) Ricketts, G., "How To Do More in Less Time," AI Expert, January, 1988, pp. 46-52.

(30) Robertson, P., "OPSAda3: An Ada-based Production Rule Interpreter," Proceedings of the Second Washington Ada Symposium, 24-26 March, 1985, pp. 139-146.

(31) Rude, A., "Translating a Research LISP Prototype to a Formal Ada Design Prototype," Proceedings of the Second Washington Ada Symposium, 24-26 March, 1985, pp. 147-170.

(32) Schleher, D., Introduction to Electronic Warfare, subchapter 8.7, "Artificial Intelligence,", Artech House, 1986, pp. 544-548.

(33) Schwan, K., Bihari, T., Weide, B., and Taulbee, G., "High Performance Operating System Primitives for Robotics and Real-Time Control Systems," ACM Transactions on Computer Systems, Vol. 5, No. 3, August 1987, pp. 189-231.

(34) Schwarz, J., "Optimization of very high level languages," Journal of Computer Languages, part 1: Vol. 1, No. 2, pp. 161-194, part 2: Vol. 1, No. 3, pp. 197-218, 1975.

(35) Schwartz, R., and Melliar-Smith, P., "The Suitability of Ada for Artificial Intelligence Applications," SRI International unnumbered technical report, May 1980 (ARO contract #AAG29-79-C-0216).

(36) Smith, D., "ALEXI - A Case Study in Design Issues for LISP Capabilities in Ada," Proceedings of the Fifth Washington Ada Symposium, 27-30 June, 1988, pp. 109-116.

(37) Stankovic, J., "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems," IEEE Computer, Vol. 21, No. 10, October 1988, pp. 10-19.

(38) Wright, M., Green, M., Feigl, G. and Cross, P., "An Expert System for Real-Time Control," IEEE Software, Vol. 3, No. 2, March 1986, pp. 16-24.

(39) Zhao, W., Ramamritham, K., and Stankovic, J., "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," IEEE Transactions on Software Engineering, Vol. SE-13, No. 5, May 1987, pp. 564-577.